Algoritmid ja andmestruktuurid

ja nende keerukus

Marko Kääramees

Miks on vaja algoritme ja andmestruktuure

Leida kiireim tee punktist A punkti B

- Andmestruktuurid

  Andmed tuleb kuidagi esitada

  – Sõidurajad, võimalikud pöörded, mitmetasandilised teed, ühesuunalised tänavad, liikumiskiiruse erinevus

- Kiire algoritm
  – Piiratud arvutusvõimsus
  – Suur andmemaht (kõik Euroopa teed-tänavad)
  – Arvuti ei vaata kaardile “ülevalt-alla” arvutada tuleb andmebaasi kirjetel, mitte graafilisel pildil

Kuidas sorteerida mullitajaid?

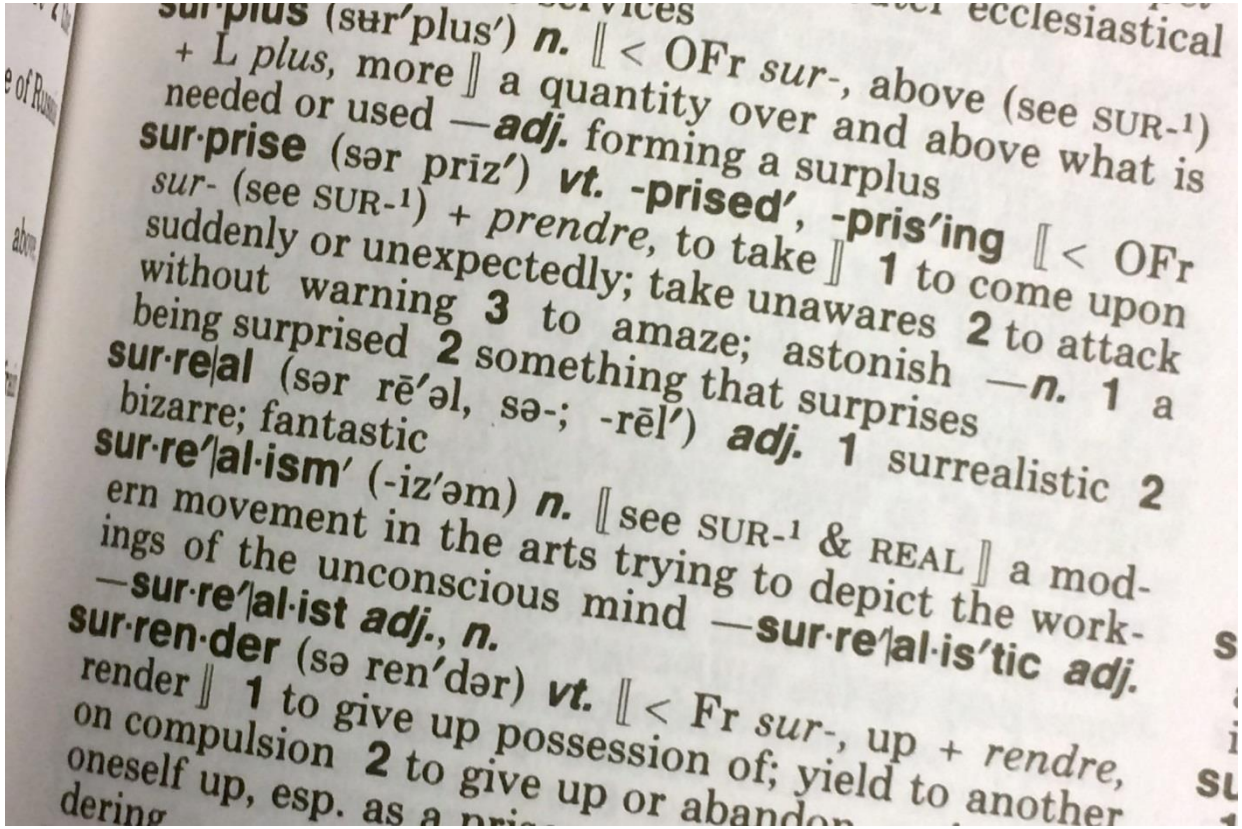
Reeglid:

See mullitaja mida kätte võtad, tuleb panna kaalule

Võidab see, kes kaalub kõige vähem kordi

Miks sorteerida?

- Otsimine põhineb sellel, et andmed on sorteeritud



sur·plus (sʉr′plus′) ***n.*** ‖ < OFr *sur-*, above (see SUR-1) + L *plus*, more ‖ a quantity over and above what is needed or used —***adj.*** forming a surplus

sur·prise (sər priz′) ***vt.*** **-prised′**, **-pris′ing** ‖ < OFr *sur-* (see SUR-1) + *prendre*, to take ‖ **1** to come upon suddenly or unexpectedly; take unawares **2** to attack without warning **3** to amaze; astonish —***n.*** **1** a being surprised **2** something that surprises

sur·re|al (sər rē′əl, sə-; -rēl′) ***adj.*** **1** surrealistic **2** bizarre; fantastic

sur·re′|al·ism′ (-iz′əm) ***n.*** ‖ see SUR-1 & REAL ‖ a modern movement in the arts trying to depict the workings of the unconscious mind —**sur·re′|al·is′tic** ***adj.*** —**sur·re′|al·ist** ***adj.***, ***n.***

sur·ren·der (sə ren′dər) ***vt.*** ‖ < Fr *sur-*, up + *rendre*, render ‖ **1** to give up possession of; yield to another on compulsion **2** to give up or abandon oneself up, esp. as a

Selection sort

Korda kuni kõik on sorteeritud

leia minimaalne listi sorteerimata osast

tõsta see listis järgmise sorteerimata kohale

```
SelectionSort(list):
list : array of items
n : size of list
for i = 1 to n - 1
    min = i
    for j = i+1 to n
        if list[j] < list[min] then
            min = j;
if indexMin != i then
    swap list[min] and list[i]
```

Mitu võrdlust on vaja?

- List 4 elemendiga
  - Min 4 elemendist – 3 võrdlust
  - Min 3 elemendist – 2 võrdlust
  - Min 2 elemendist – 1 võrdlust
  - Kokku 3+2+1 = 6 võrdlust
- List 8 elemendiga
  - Kokku 7+6+5+4+3+2+1 = 28 võrdlust
- Üldjuhul *n* elemendi korral aritmeetiline jada *n*-1

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2}$$

3	1	4	2
1	3	4	2
1	2	4	3
1	2	3	4

n	võrdlusi
4	6
8	28
16	120
32	496

Algoritmide keerukus

Algoritmi keerukus on põhioperatsiooni(de) arvu sõltuvusfunktsioon K(*n*) sisendi(te) suurusest *n*.

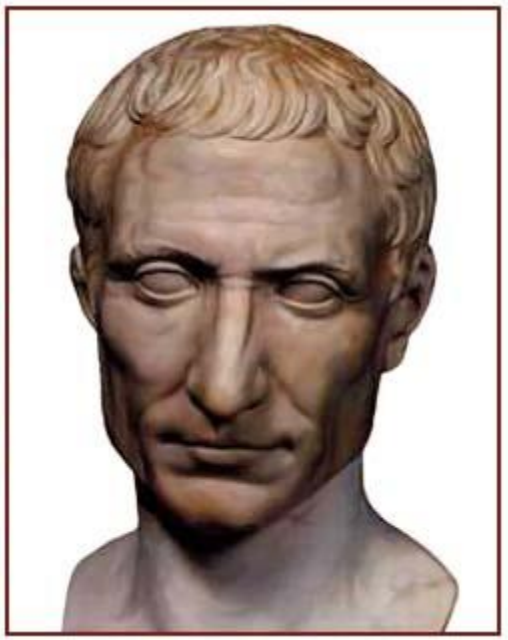
- *Põhioperatsioon* on midagi, mis on riistvaras tehtav piiratud arvu sammudega
  - aritmeetika tehe, võrdlus, omistus
  - rida programmikoodis, mis ei sisalda tsüklit ega funktsiooni
- *Sisendi suurus* võib olla defineeritud erinevalt
  - Sisendandmete maht (massiivi, listi, andmebaasi suurus)
  - Sisendparameetri väärtus
  - Sisendparameetri suurus (bittide/baitide arv)

Kas õnnestuks sorteerida vähema arvu võrdlemistega?
Efektiivsemalt?

Jaga ja Valitse strateegia

Divide et Impera

CAESAR



Napoleon

Jaga ja Valitse strateegia

❶ Jaga – jaga alamülesanneteks

❷ Valitse – lahenda alamülesanded

❸ Ühenda – kasuta alamülesannete lahendusi

Merge sort

❶ Jaga – jaga alamülesanneteks

Jagame andmed kahte massiivi pooleks

❷ Valitse – lahenda alamülesanded

Sorteerime mõlemad pooled eraldi

Triviaalse alamülesande lahendus (rekursiooni lõpp):

Ühe elemendiga massiiv

Tagastame, kuna on juba sorteeritud

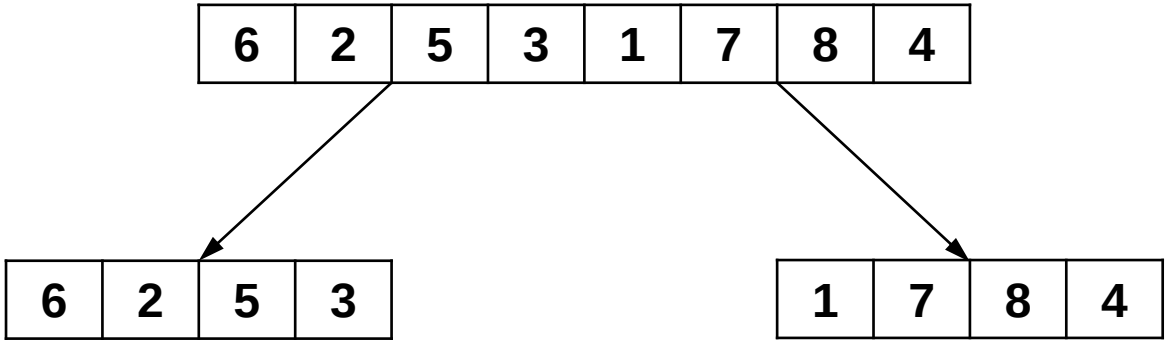
❸ Ühenda – kasuta alamülesannete lahendusi

Ühendame sorteeritud massiivid üheks sorteeritud massiiviks võrreldes omavahel mõlema massiivi väiksemaid elemente

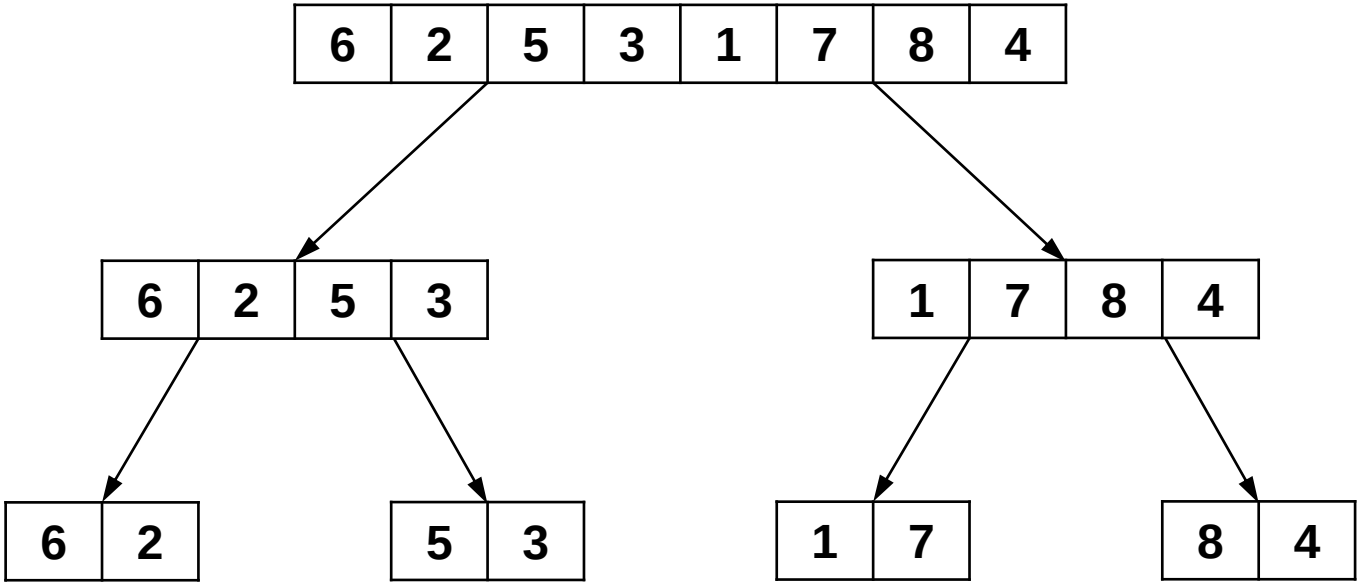
Merge sort

6	2	5	3	1	7	8	4

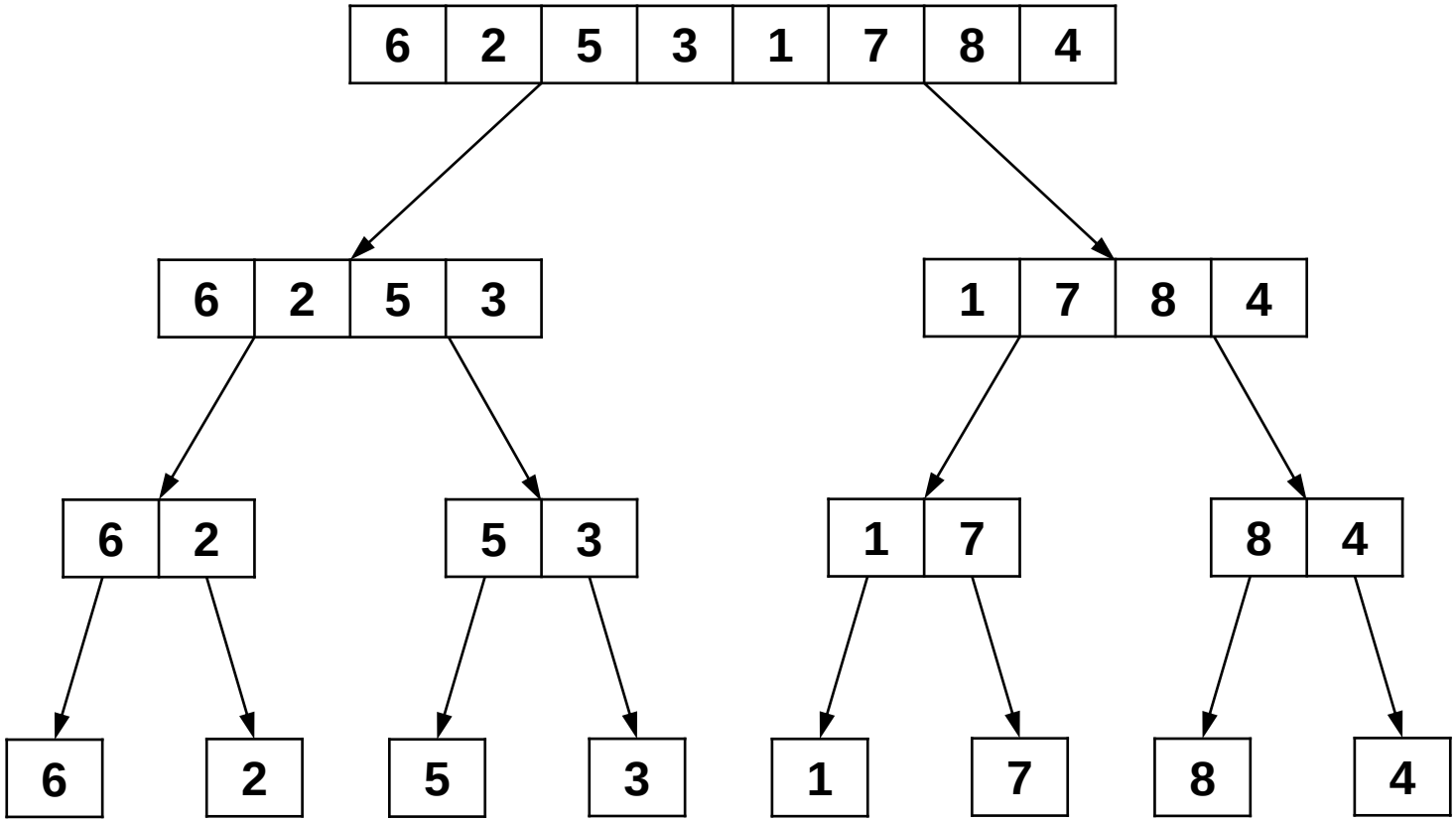
Merge sort



Merge sort

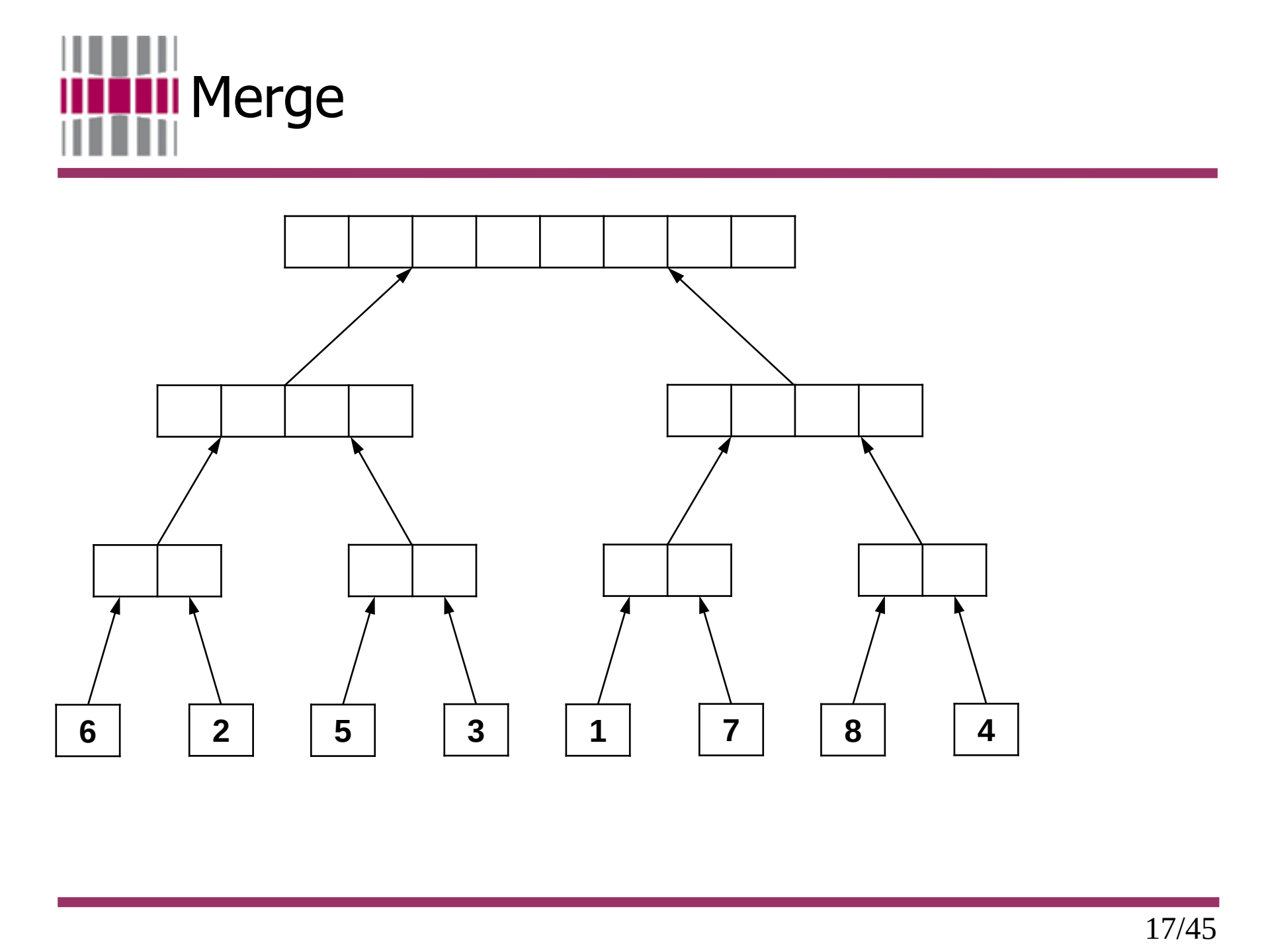


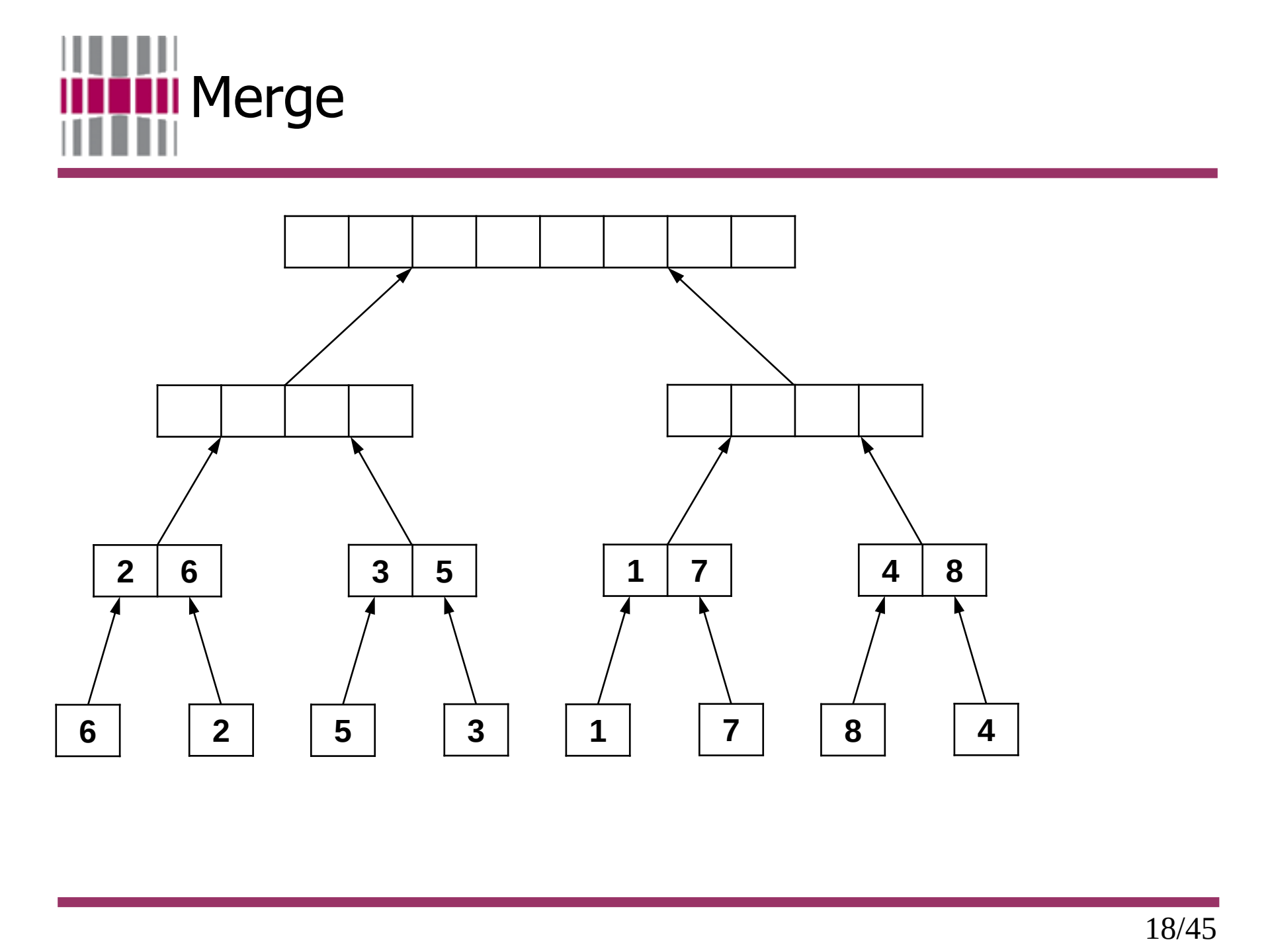
Merge sort



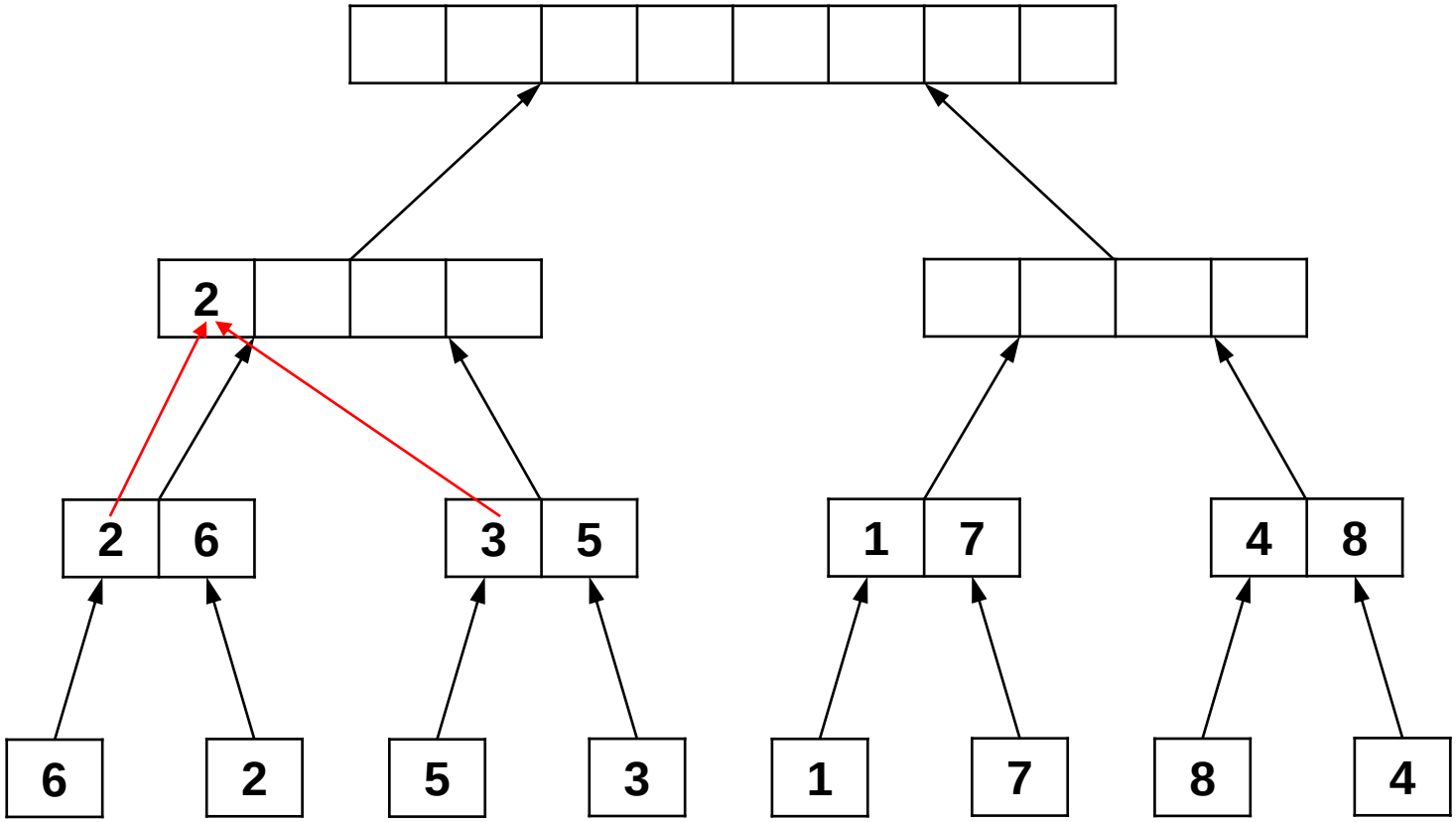
Merge sort

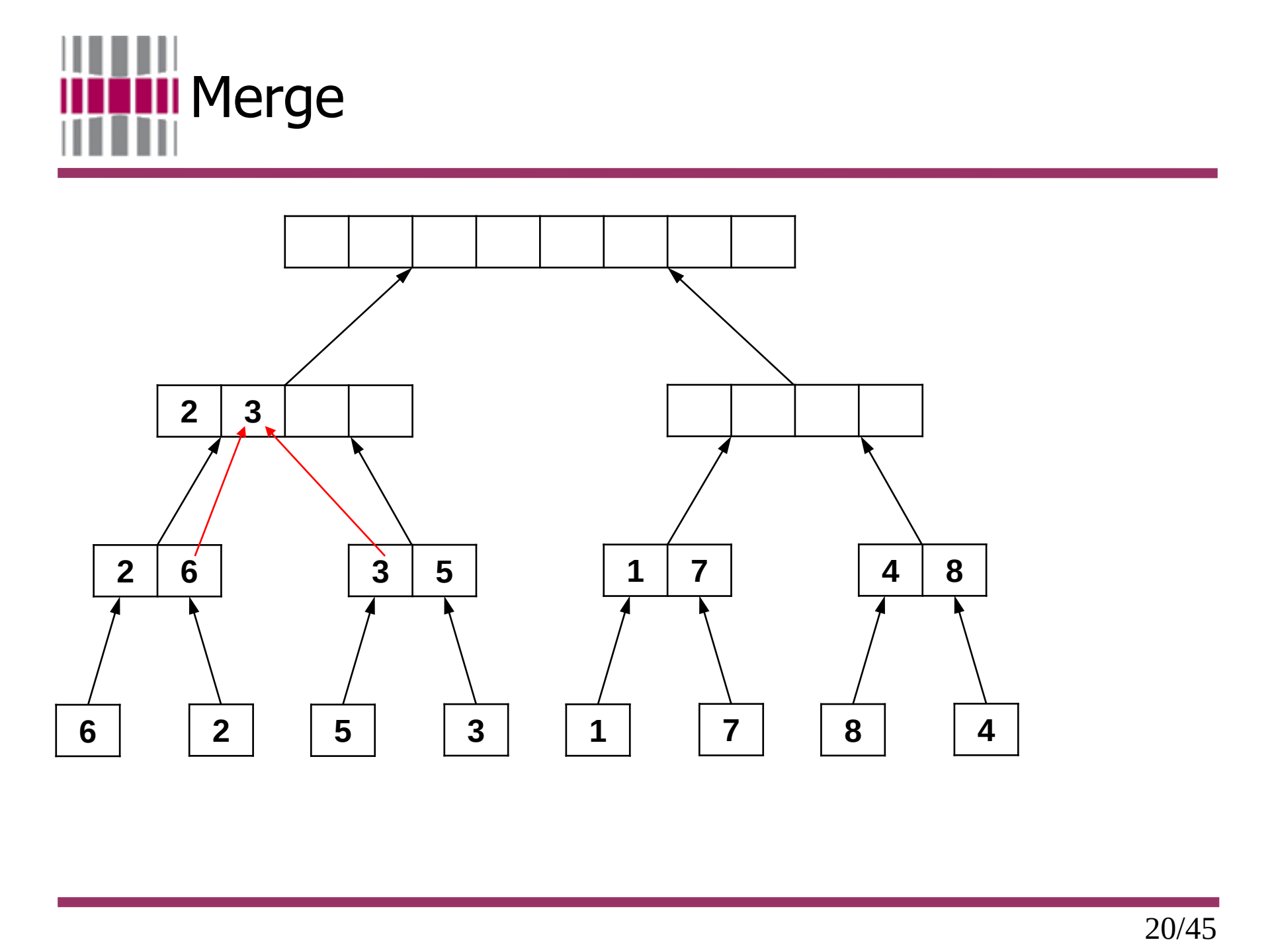
```
mergesort (int n, list S)
   if (n>1)
      h=n/2,        m = n - h
      list U[1 ..h], V[1 .. m]
      copy S[1] through S[h] to U[1] through U[h]
      copy S[h+1] through S[n] to V[1] through V[m]
      mergesort(h, U)
      mergesort(m, V)
      merge (h, m, U, V, S)
```

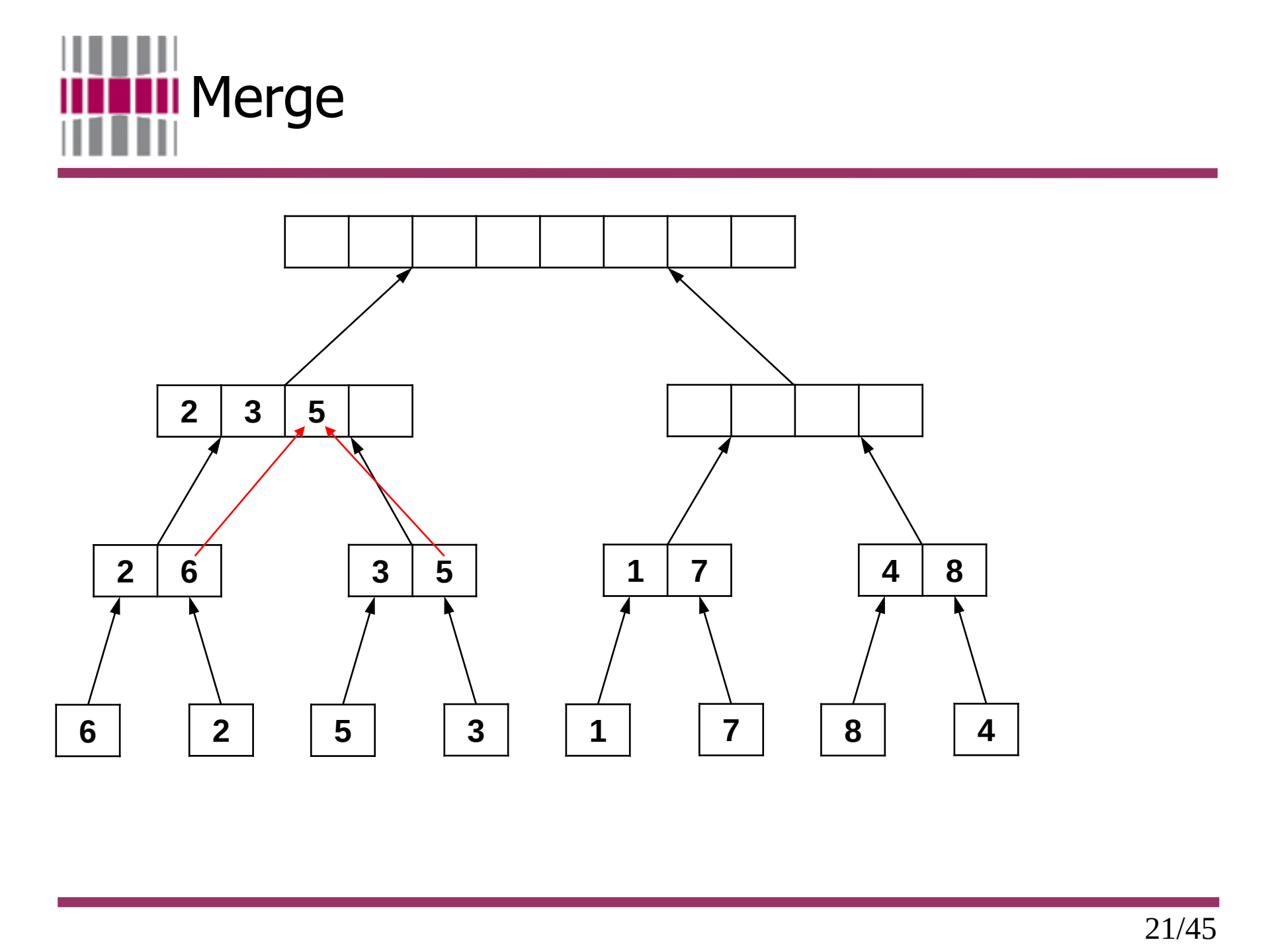

Merge
6
2
5
3
1
7
8
4


Merge
2
6
3
5
1
7
4
8
6
2
5
3
1
7
8
4

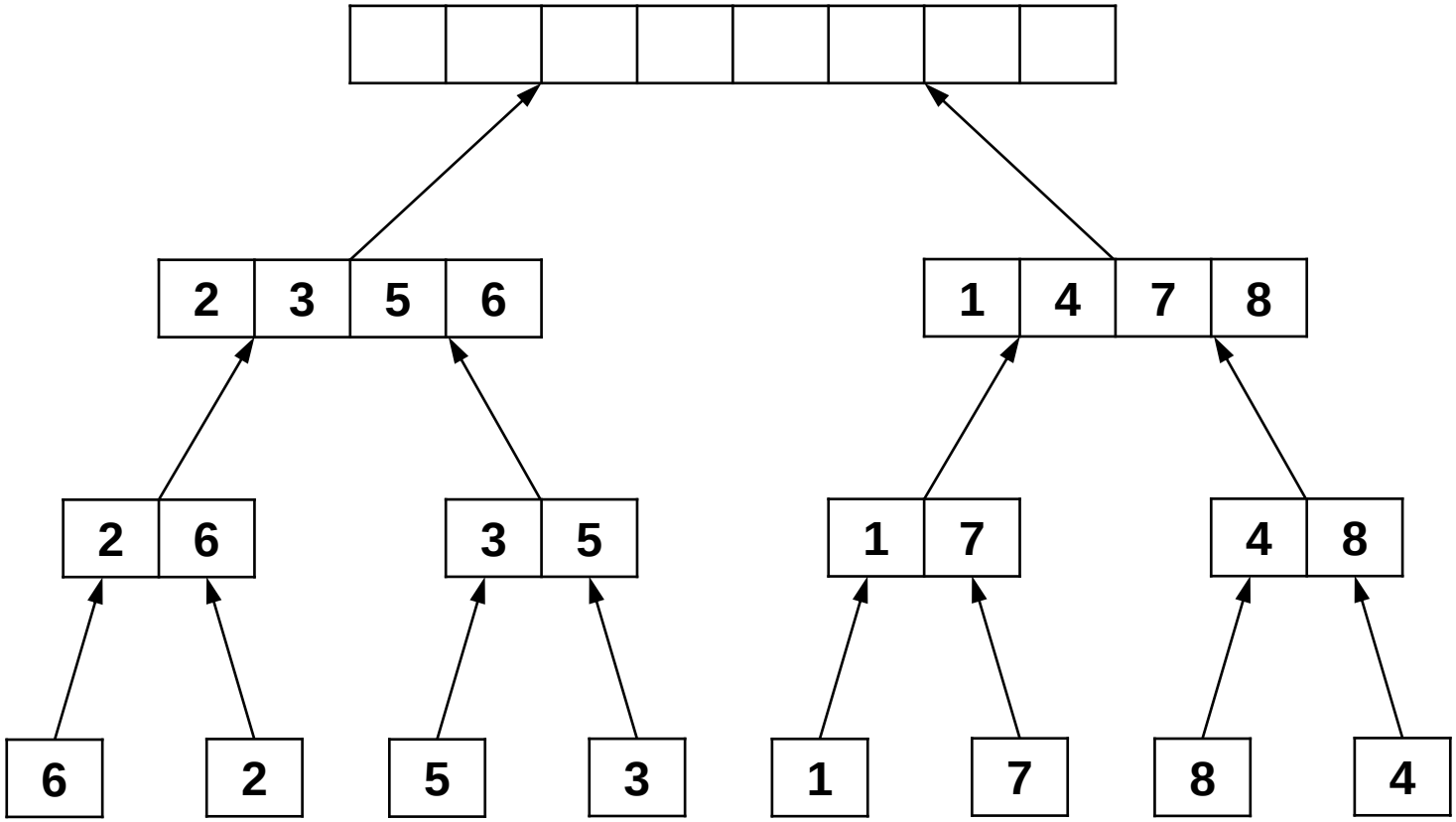
Merge



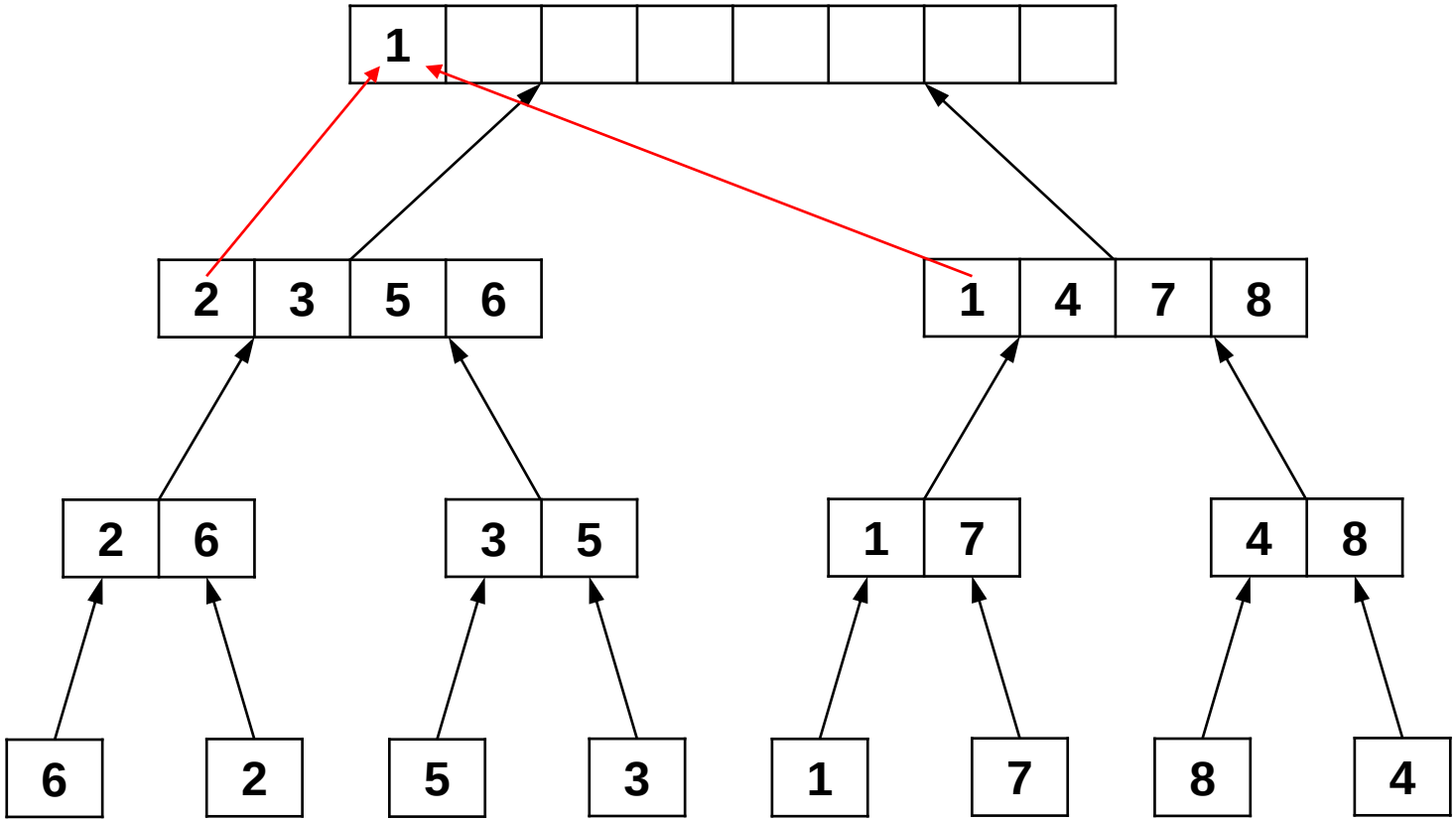

Merge
2
3
2
6
3
5
1
7
4
8
6
2
5
3
1
7
8
4


Merge
2
3
5
2
6
3
5
1
7
4
8
6
2
5
3
1
7
8
4

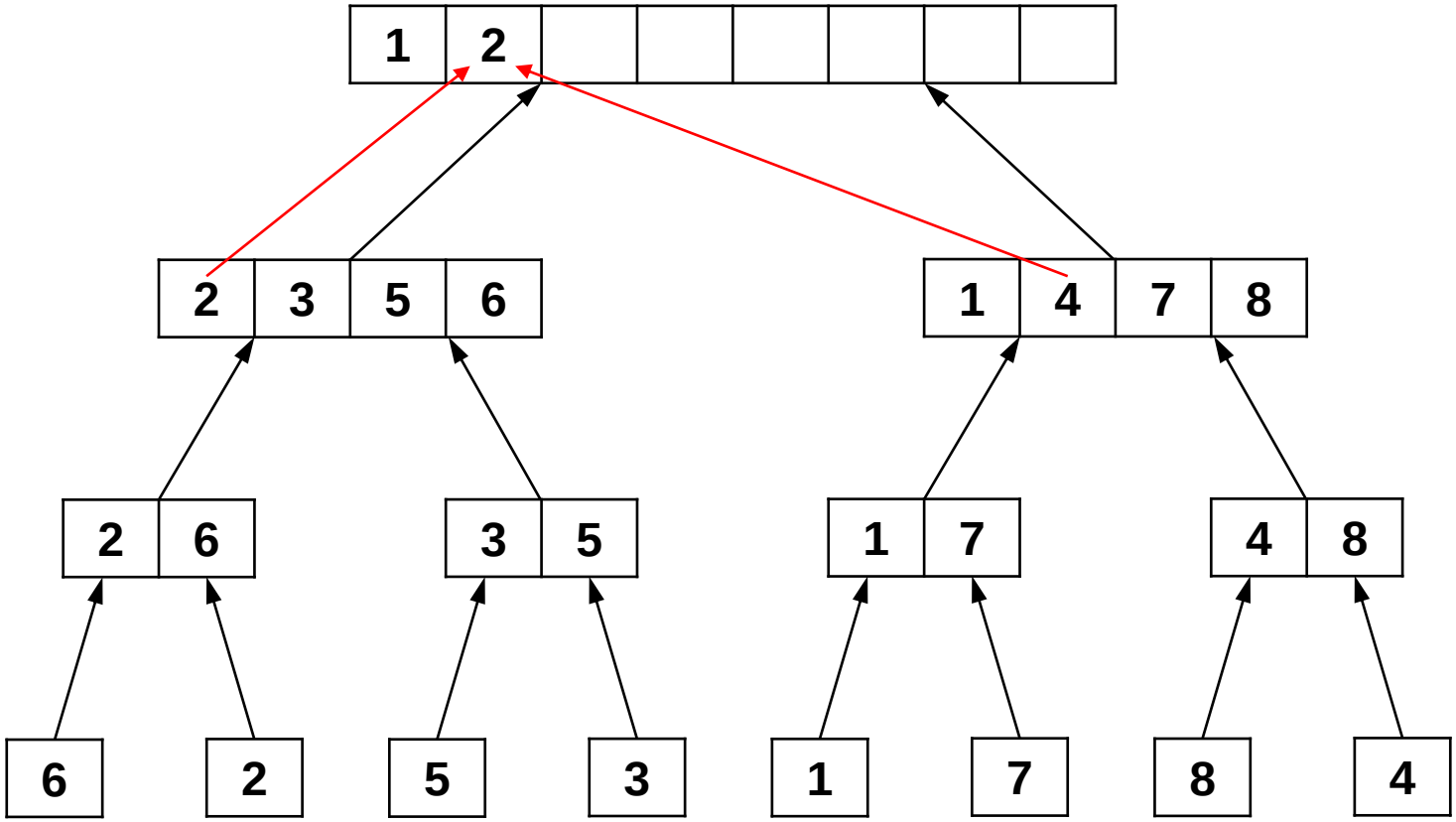
Merge

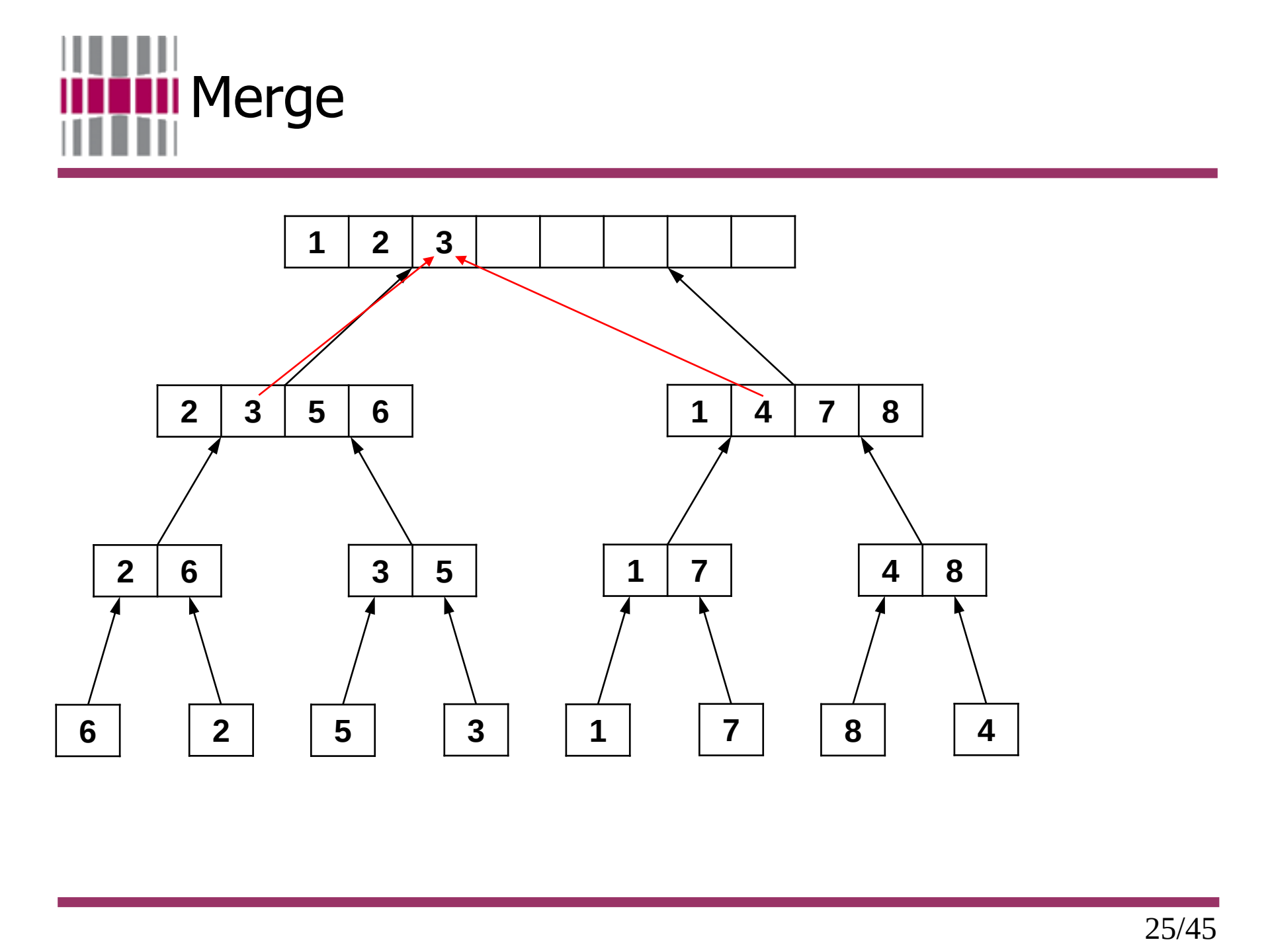


Merge

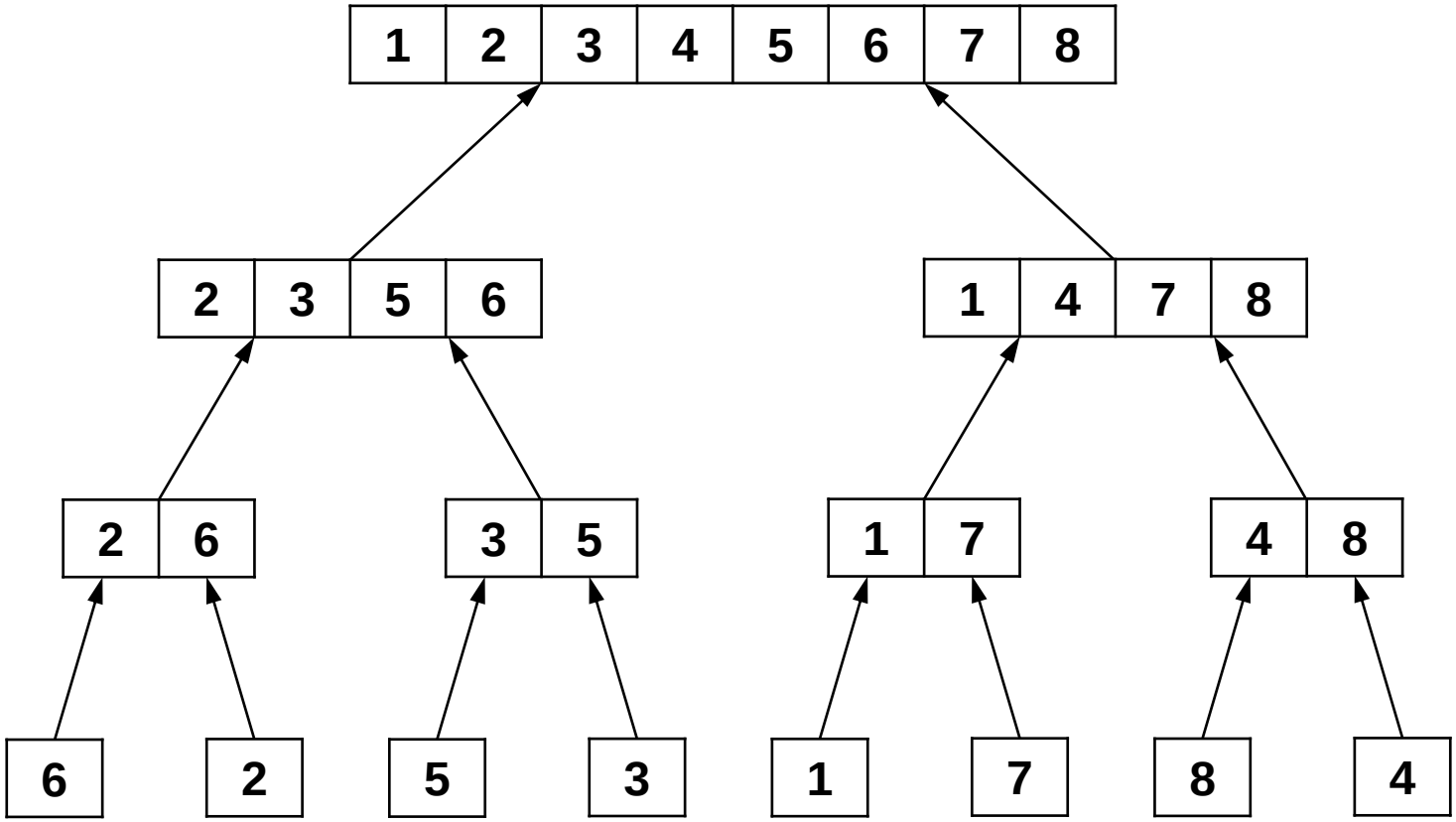


Merge




Merge
1 2 3
2 3 5 6
1 4 7 8
2 6
3 5
1 7
4 8
6
2
5
3
1
7
8
4

Merge

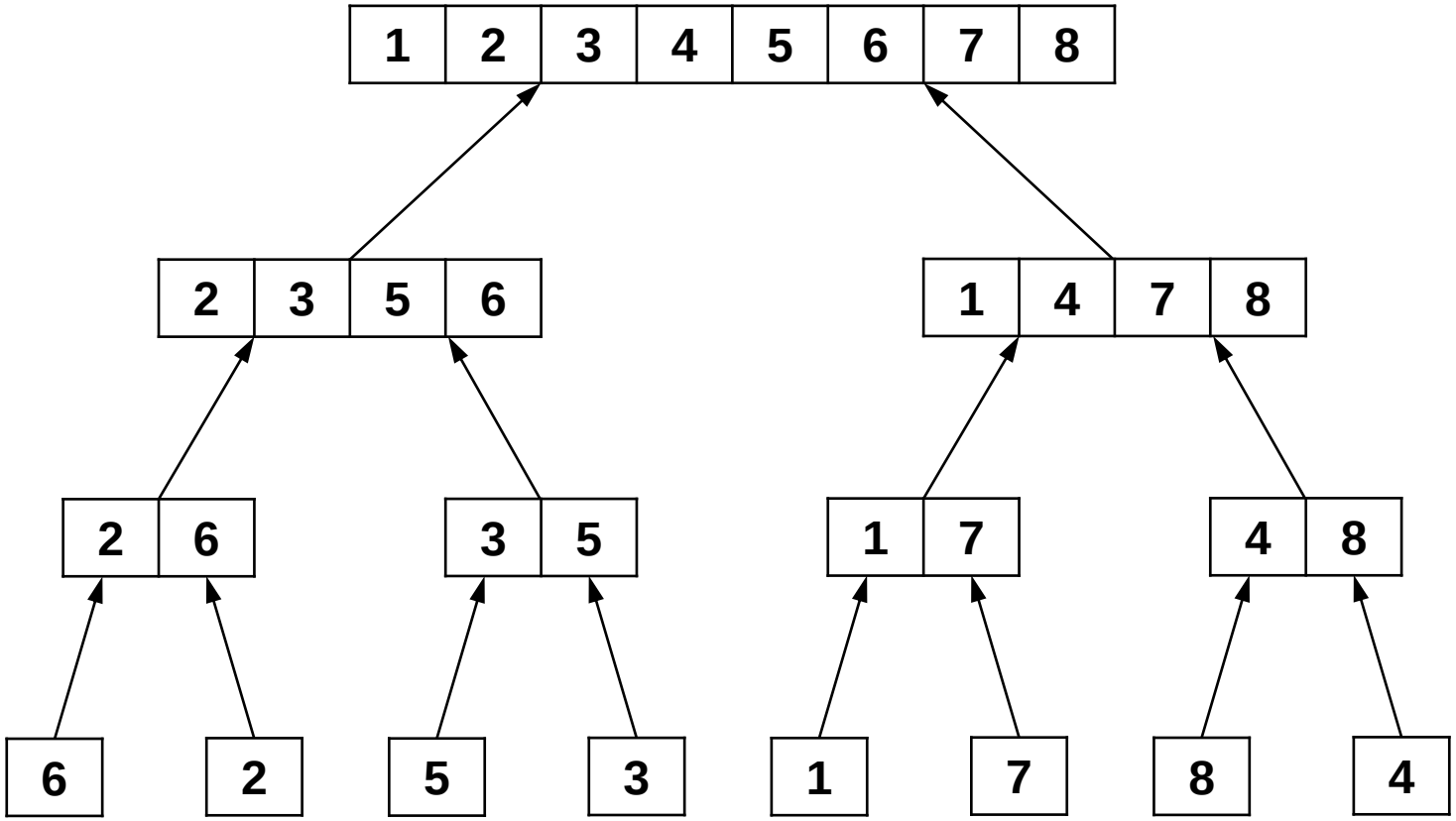


Merge sort

```
mergesort (int n, list S)
   if (n>1)
      h=n/2,        m = n - h
      list U[1 ..h], V[1 .. m]
      copy S[1] through S[h] to U[1] through U[h]
      copy S[h+1] through S[n] to V[1] through V[m]
      mergesort(h, U)
      mergesort(m, V)
      merge (h, m, U, V, S)
```

```
merge (int h, int m, list U, list V, list S)
   int i, j, k
   i = 1; j = 1; k = 1
   while (i <= h && j <= m)
      if (U[i] < V[j])
         S[k] = U[i]; i++
      else
         S[k] = V[j]; j++
      k++
   if (i>h)
      copy V[j] through V[m] to S[k] through S[h+m];
   else
      copy U[i] through U[h] to S[k] through S[h+m];
```

Mitu võrdlust on vaja?



1*7 = 7

2*3 = 6

4*1 = 4

Kokku: 7+6+4 = 17

Üldjuhul kui n=2^k võrdlusi: $(n\text{-}1)\ log_2(n\text{-}1) + log_2(n)$

Mis vahet neil on?

Eestis oli elanikke 1. jaanuar 2017

1315635

- Sorteerime need ära
  - Arvuti protsessor töötab 4 GHz
  - Iga 10 taktiga tehakse üks võrdlus

Algoritm	Võrdlusi üldjuhul	Võrdlusi	Aega
Selection sort	$\frac{1}{2}(n^2-n)$	$8.7*10^{11}$	2180s = 35 min
Merge sort	$(n-1)\ log_2(n-1) + log_2(n)$	$2.7*10^7$	0.07s

O-notatsioon

- Annab keerukusklassi – millise proportsiooniga suureneb arvutusaeg sõltuvalt sisendi suuruse muutusest
- O-notatsioonis esitatakse keerukusfunktsiooni määrav komponent

$$1/2\ (n\text{-}1)*n \in O(n^2)$$

- Liidetavatest suurima keerukusega funktsioon

$$n^2 - n \in O(n^2)$$

- Konstantseid kordajaid võib ignoreerida

$$½*n^2 \in O(n^2)$$

- Spetsiaalsed keerukusklassid:

logaritmiline:	**O(log *n*)**
lineaarne:	**O(*n*)**
polünomiaalne:	**O(n^k), $k \geq 1$**
eksponentsiaalne:	**O(a^n), $n > 1$**

Sorteerimiste keerukused

- Selection sort

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} \in O(n^2)$$

- Merge sort

$(n-1)\log(n-1) + \log(n) \in O(n \log n)$

Neid on võimalik määrata ilma katsetamata, algoritmi analüüsides

Quicksort

❶ Jaga – jaga alamülesanneteks

Valime ühe elemendi teljeks (pivot)

Sorteerime kõik teljest väiksemad temast vasakule ja suuremad paremale

❷ Valitse – lahenda alamülesanded

Sorteerime teljest paremale ja vasakule jääva massiivi

Triviaalse alamülesande lahendus (rekursiooni lõpp):

Ühe elemendiga massiiv

Tagastame, kuna on juba sorteeritud

❸ Ühenda – kasuta alamülesannete lahendusi

Ühendame vasaku massiivi, telje ja parema massiivi üksteise järgi

Erinevad keerukusmõõdud

n	Võrdlusi min	Võrdlusi max
2	1	1
4	4	6
8	13	28

- Halvima juhu keerukus $O(n^2)$
- Keskmise juhu keerukus $O(n \log n)$

Mis on *quicksort-i* jaoks halvim sisend?

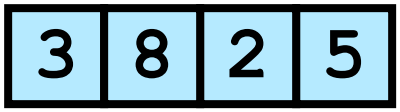
Edasi?

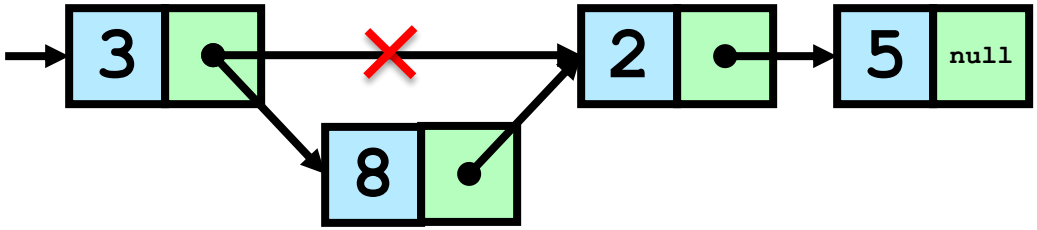
- Kas saab veel kiiremini?
- Kui kasutada võrdlemist, siis ei saa
- Vaatame mis tingimustel saab

Andmestruktuurid

- Sisseehitatud andmetüübid
  - Täisarv, ujukomaarv
  - String
- Andmestruktuur
  - List, Set
  - Queue, Stack
  - Dictionary
- Andmestruktuur võimaldab andmeid hoida ja eesmärgipäraselt ning lihtsalt kasutada
  - Liides – meetodid/funktsioonid töötamiseks
  - Implementatsioon – optimiseeritud valitud funktsioonidele

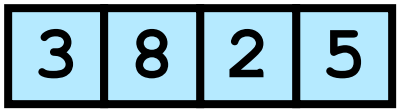
Massiiv (Python list) ja lingitud list

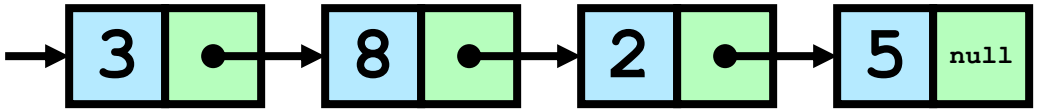




Funktsioon	**Massiiv**	**Lingitud list**
Lisa lõppu	O(1)	O(1)
Lisa algusesse	O(n)	O(1)
Otsi i-s element	O(1)	O(n)
Kustuta element	O(n)	O(1) O(n) koos otsinguga

Massiiv (Python list) ja lingitud list





Massiiv

+ kiire otsepöördumine
- aeglane *add/delete* algusesse või keskele

LinkedList

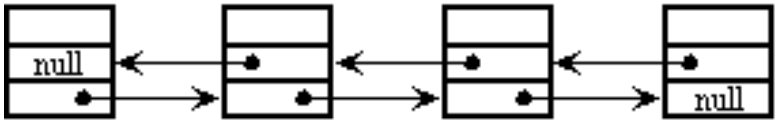
- aeglane otsepöördumine
+ kiire *add/delete*

Lingitud andmestruktuurid

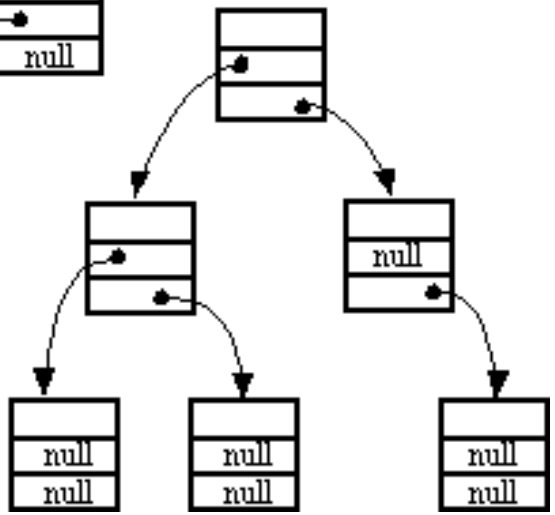
```
class Node {
  Data item;
  Node next;
}
```

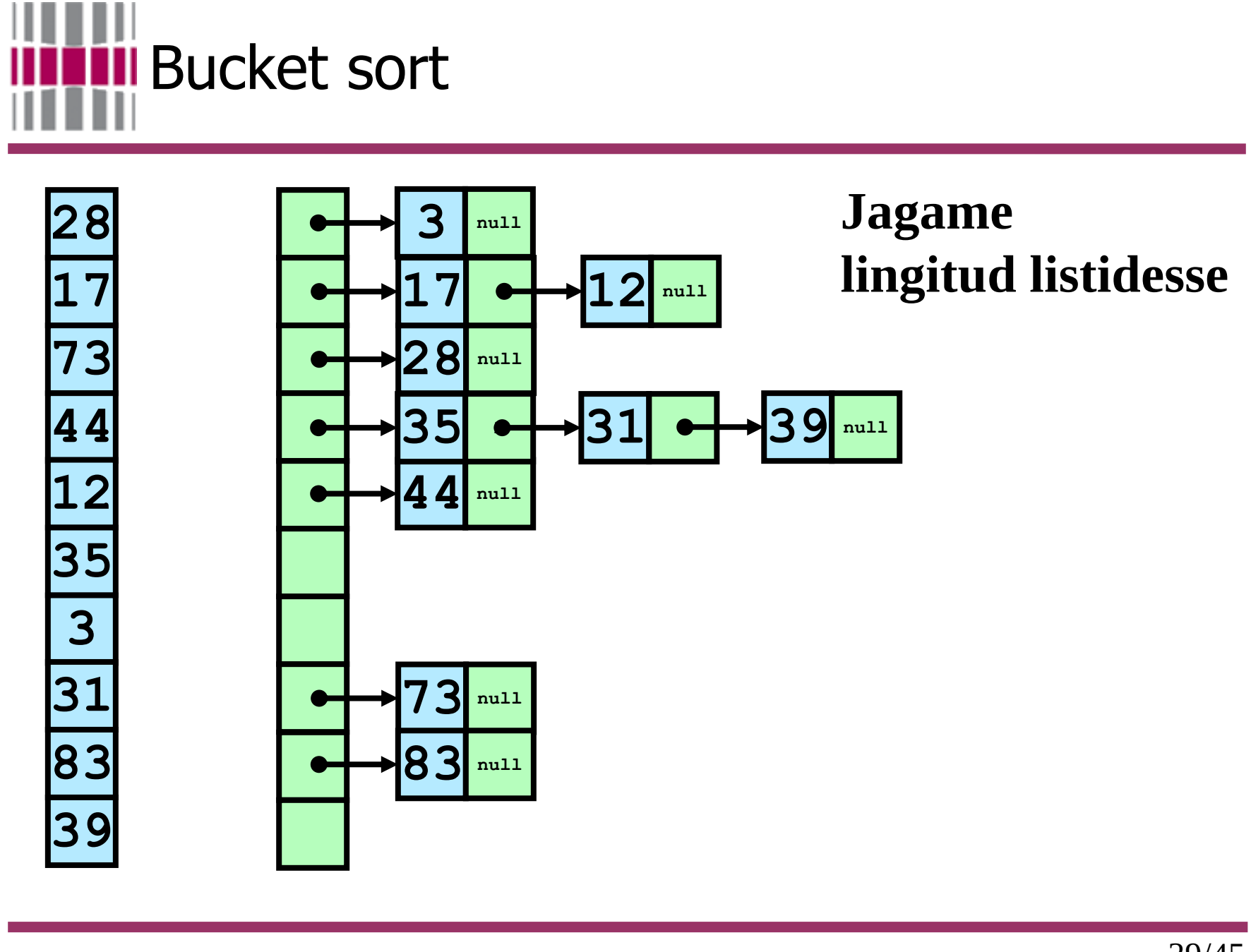
When an object contains a reference to an object of the same type, then several objects can be linked together into a list. Each object in the list refers to the next.

```
class Node {
  Data item;
  Node left;
  Node right;
}
```

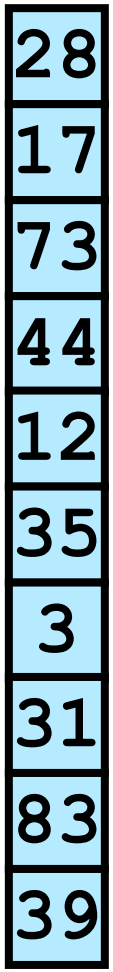


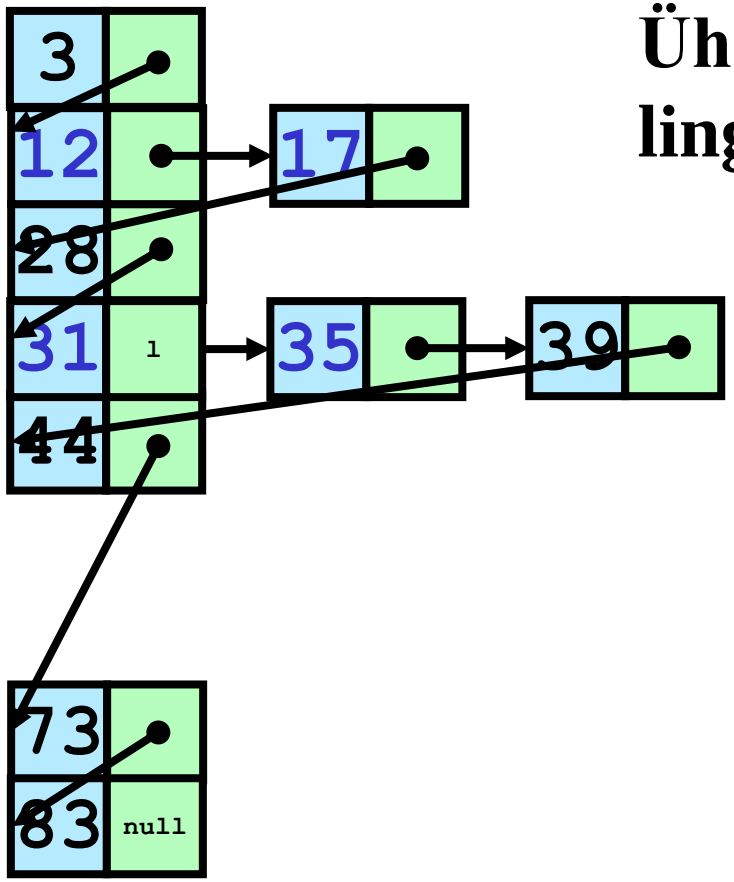
Things get even more interesting when an object contains two references to objects of the same type. In that case, more compliated data structures can be constructed.




Bucket sort
28
17
73
44
12
35
3
31
83
39
3
null
17
12
null
28
null
35
31
39
null
44
null
73
null
83
null
Jagame
lingitud listidesse

Bucket sort





Ühendame lingitud listid

Bucket Sort

Bucket-Sort(A, x, y)

1. Jaga vahemik [x, y) n-*ks* võrdseks alamvahemikuks (buckets)
2. Jaga n sisendvõtit alamvahemikesse (*buckets*)
3. Sorteeri iga alamvahemik (kasutades näiteks insertion sort-i)
4. Üheda (sorteeritud) alamvahemikud (bucket) järjekorras kokku väljundiks

Keerukus: O(n)

1: O(1) iga vahemiku kohta = O(n) kokku.

2: O(n).

3: Eeldatav bucket suurus on O(1), kokku O(n)

4: O(1) iga vahemiku kohta = O(n) kokku

Mis vahet neil on?

n = **1315635** sorteeritavat

Sorteerime need ära

- Arvuti protsessor töötab 4 GHz taktsagedusel
- Iga 10 taktiga tehakse üks põhioperatsioon

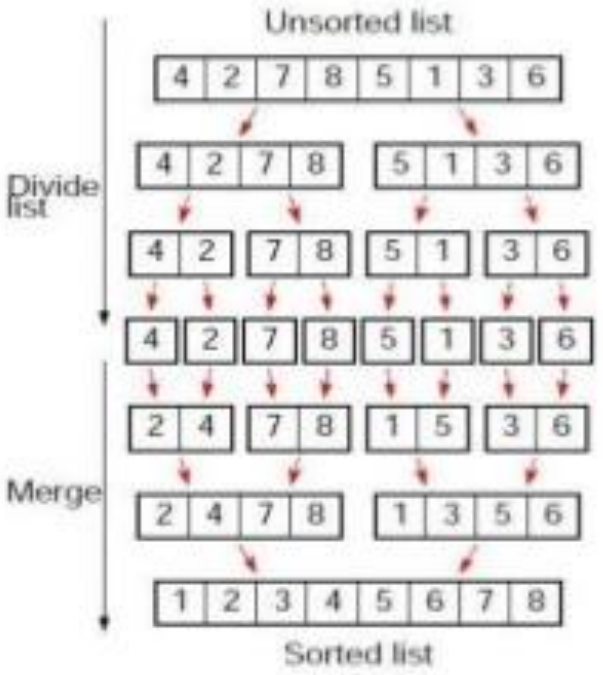
Algoritm	Võrdlusi üldjuhul	Võrdlusi	Aega
Selection sort	$\frac{1}{2}(n^2-n)$	$8.7*10^{11}$	2180s = 35 min
Merge sort	$(n-1)\ log_2(n-1) + log_2(n)$	$2.7*10^7$	0.07s
Bucket sort	n	$1.3*10^6$	0.003s

Bucketsort omadused

- Keerukus O(*n*)
- Kiire
- Eeldab, et sorteeritavad arvud on ühtlase jaotusega
  - Igasse vahemikku (*bucket*) tulevad mõned üksikud arvud
- Isikukoodid ei ole ühtlase jaotusega
  - Seda sorteerimist ei ole otstarbekas igas olukorras kasutada

Paralleelne Mergesort

```
ALGORITHM: mergesort(A)
1  if  (|A| = 1) then return A
2  else
3       in parallel do
4            L := mergesort(A[0..|A|/2))
5            R := mergesort(A[|A|/2..|A|))
6       return merge(L, R)
```



Keerukus O(n), vajab O(n) protsessorit

Odd-Even Parallel Mergesort võib sorteerida keerukusega O((log n)^2), vajab O(n) protsessorit

Kokkuvõtteks

- **Tasub mõelda**
  probleemi lahendades tasub leida efektiivne algoritm
- **Tasub analüüsida**
  Keerukust saab analüüsida algoritmi pealt ilma programmeerimata ja aega mõõtmata
- **Vee all võivad olla karid**
  Alati ei lange keskmine ja halvima juhu keerukus kokku
- **Mõned algoritmid toimivad ainult kindlatel eeldustel**
- **Jõuga ja nõuga võib saada rohkem**
  Ülesande ümberdefineerimine ja paralleelsus aitavad